Towards Certified Model Checking for PLTL Using One-Pass Tableaux

Alex Abuin
Ikerlan Technology Research Centre,
P. J. M. Arizmendiarrieta, 2 20500 Arrasate-Mondragón Gipuzkoa, Spain
aabuin@ikerlan.es

Alexander Bolotov
University of Westminster,
W1W 6UW, London, UK
https://www.westminster.ac.uk/about-us/our-people/directory/bolotov-alexander
A.Bolotov@westminster.ac.uk

Unai Díaz de Cerio
Ikerlan Technology Research Centre,
P. J. M. Arizmendiarrieta, 2 20500 Arrasate-Mondragón Gipuzkoa, Spain
udiazcerio@ikerlan.es

Montserrat Hermo
University of the Basque Country,
P. Manuel de Lardizabal, 1, 20018-San Sebastián, Spain
montserrat.hermo@ehu.es

Paqui Lucio
University of the Basque Country,
P. Manuel de Lardizabal, 1, 20018-San Sebastián, Spain
http://www.sc.ehu.es/paqui
paqui.lucio@ehu.eus

Abstract

The standard model checking setup analyses whether the given system specification satisfies a dedicated temporal property of the system, providing a positive answer here or a counter-example. At the same time, it is often useful to have an explicit proof that certifies the satisfiability. This is exactly what the *certified model checking (CMC)* has been introduced for. The paper argues that one-pass (context-based) tableau for PLTL can be efficiently used in the CMC setting, emphasising the following two advantages of this technique. First, the use of the context in which the eventualities occur, forces them to fulfil as soon as possible. Second, a dual to the tableau sequent calculus can be used to formalise the certificates. The combination of the one-pass tableau and the dual sequent calculus enables us to provide not only counter-examples for unsatisfied properties, but also proofs for satisfied properties that can be checked in a proof assistant. In addition, the construction of the tableau is enriched by an embedded solver, to which we dedicate those (propositional) computational tasks that are costly for the tableaux rules applied solely. The combination of the above techniques is particularly helpful to reason about large (system) specifications.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Temporal logic, fairness, expressiveness, linear-time, Certified model checking

Digital Object Identifier 10.4230/LIPIcs.TIME.2019.12

Funding *Alexander Bolotov*: This author has been partially supported by the UK Knowledge Transfer Partnership KTP011063 Lumina Learning & University of Westminster, and the Spanish Project TIN2017- 86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU18-182.
Montserrat Hermo: This author has been partially supported by Spanish Project TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU18-182.
Paqui Lucio: This author has been partially supported by Spanish Project TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU18-182.

26th International Symposium on Temporal Representation and Reasoning (TIME 2019).
Editors: Johann Gamper, Sophie Pinchinat, and Guido Sciavicco; Article No. 12; pp. 12:1–12:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction and Problem Setup

Model Checking [10, 31, 11] is an algorithmic method for determining whether a complex hardware or software system satisfies a given property. Many important properties to be verified reflect the system's dynamics and are expressed in some temporal logic. If the property does not hold, the checker returns a counter-example: a trace/model of the system that does not satisfy the property. This counter-model acts as a "certificate" of the failure and its role is to help the user to identify the source of the problem which could be in the system design, in the property, and even in the model checker. However, there are a number of scenarios where another certification is needed. One of these scenarios consists on proving, only once, the correction of the underlying algorithm of the model checker. For this task, interactive proof assistants such as Coq or Isabelle are good tools. They allow us to certify a model checker and even obtain an executable program by the refinement of some extraction mechanism. For instance Amjad [1] described how to code BDD-based symbolic model checking algorithms into an automatic theorem prover. More recently, Esparza et al. [13] have verified an automata-based model checker with Isabelle theorem prover. The second scenario involves the model checker to certify that a particular property is true. For example, the user may deal with a very complex specification and is not sure if the specification is well written. Here it is important to have techniques that provide not only a counter-example, but also certify that the system meets the property. This is exactly what the *certified model checking (CMC)* has been introduced for. In this second scenario a number of techniques have been previously proposed. Some of the techuiques that deal with finite-state systems can be found in [23, 29]. In [23] an automata-theoretic approach to model checking is addressed. In [29] a deductive proof system was introduced for verifying branching time properties expressed in the mu-calculus. For infinite-state systems, Mebsout et al. [28] recently presented a new technique for generating and verifying proof certified in SMT-based model checkers, focusing on proofs of invariant properties. The use of invariants has been exploited in [17, 22], where the proof is generated from the inductive invariant obtained with the k-liveness algorithm [9]. The resulting approach can be implemented as a model checker based on the combination of k-liveness with an engine for invariant properties that is capable of producing inductive invariants. A drawback of this approach is that, although it is very competitive, the task of finding counter-examples and the task of generating proofs (in this case via finding invariants), are very different requiring for the latter the addition of extra mechanisms to the own model checker.

In this paper, we propose an CMC based on dual systems of tableaux and sequent calculus, originally introduced in [14, 15]. It produces certificate proofs, formal proofs in the sequent calculus, and counterexamples - open branches in the tableau. This is also one of the main advantages of our approach: the same reasoning mechanism applies for both tasks - certificates and counterexamples.

The CMC (see Figure 1) differs from the traditional model checking in providing a proof of the satisfiability of the given property, and not only a counter-example. Incorporating the notation of [21] (and slightly modifying it) we represent the methodology of the certified model checking as the following signature:

$$\mathcal{CMC} :: System \times \varphi \longrightarrow \mathbb{B} \times (Proof \mid Counter\text{-}example)$$

where, given a specification of a system, S, and a property, φ, a certified model checking produces a Boolean result, $\mathbb{B}$, indicating whether S satisfies φ, along with

- a proof (or certification), in the positive case, or
- a counter-example, in the negative case.

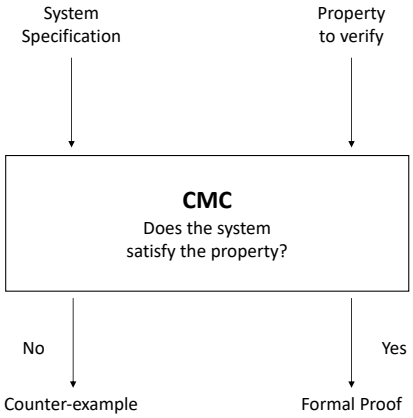

Figure 1 General schema of CMC.

However, we believe that to take the full advantage of CMC, and enabling its industrial application to real systems, we need to ensure that CMC meets the following requirements.

- **(i)** Proofs should be generated automatically.
- **(ii)** An CMC needs to offer a CMC user sufficient information to understand the proof without additional "costs" related to specialist knowledge of the underlying proof technique.
- **(iii)** The presentation of the proof should enable the CMC users to easily navigate through its trace. This becomes particularly important when the system is badly defined – here the navigation through the trace can help to detect errors.
- **(iv)** Finally, when developing a safety critical system following a safety process (e.g. processes of standard ISO26262 [19], IEC61508 [18] or EN50128 [7]), it is mandatory to analyse how a bug in a tool (a model checker in our case) may affect the safety of the developed system. Depending on the level of these effects, some actions are required to increase our confidence in the model checker. One of the mechanisms to increase the probability of finding a failure is to re-evaluate the outputs of the CMC by an independent tool developed by an independent team.

We propose a particular CMC method of realising the CMC philosophy (see Figure 2) meeting the characteristics (i)-(iv) above while maintaining the common (for the traditional Model Checker) functionality. Our method is centered on a context-based temporal one-pass tableaux technique whose performance is optimised by a SAT solver.

Various tableaux techniques have been proposed for a rich variety of temporal logics, linear and branching: *Propositional Linear-Time Temporal Logic* (PLTL); *Computation Tree Logic* (CTL); CTL* which generalizes PLTL and CTL, etc. (an excellent survey can be found in [16]). One of the core ideas of the tableaux methods is to identify eventualities within the given temporal input and to check that they are fulfilled. Traditional tableaux techniques require two phases to perform this test. In the first phase, a graph which gives all possible pre-models for the tableau input, in constructed. In the second phase, for each state, s in this graph, that contains some "eventually φ", a graph-theoretic algorithm looks for a state, reachable from s, that satisfies φ. Note also that the two-pass tableaux methods fail to maintain the classical correspondence between tableaux and sequents that associates a sequent proof with the closed tableau.

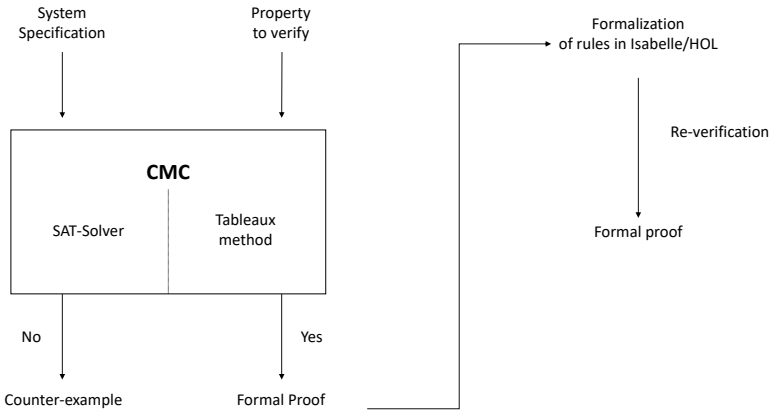

Figure 2 General schema of the proposal.

To avoid the second phase and, hence, to keep the ability of generating (sequent) proofs from tableaux, in [14, 15], dual systems of tableaux and sequents, for PLTL, were presented. Every logic defined in [14, 15] was proved to be sound and complete. In particular, [15] contains a proof that the tableau system we use in the present paper, is a decision method for the full PLTL, i.e. it is sound, refutationally complete, and terminating. The termination property is achieved on the basis of any fair selection strategy. A very similar sequent calculus is presented in [6] (see [15] for more details about the similarities and differences of these systems). The one-pass tableau method [15] has been extended to a concurrent constraint logic in [12], and also to ECTL$^\sharp$ - a branching-time sublogic of CTL* in [5].

The tableau method in [15] makes use of the so-called *context* of an eventuality to force its fulfillment. The context of an eventuality is simply the set of formulae that "accompanies" the eventuality in the label of the node. When a one-pass tableau checks whether a property φ holds or not, it is always able to issue a certificate: either a counter-model or a complete explanation (formal proof) of why φ is true.

We abbreviate the one-pass tableau method as $\tau^{\uparrow}_{pltl}$ and its dual sequent calculus as TTC.

Considering one-pass tableau method as one of the core components of the CMC solution we present in this paper, we argue that it conforms with the properties (i)-(iv) mentioned above. Indeed, the method automatically checks whether the specification of the system satisfies the property (i).[1] If it does not then it provides a counter-example. A counter-example is given by a trace which is intuitively clear (ii). In the case of a positive answer, a relevant proof certifies it. Again, the output proof is represented as a trace enabling an easy navigation through it (iii). Moreover, one of the attractive features of our method is that it forces the eventualities to be fulfilled as soon as possible, thus, the method will potentially generate shorter paths (with fewer nodes) than those produced by non-context based tableaux.

In our proposal, the performance of the one-pass tableau technique is complemented, for efficiency, by the SAT solving. Note that the idea of encoding of transition systems into propositional SAT was first proposed in [20] for AI planning problems, where the authors show that SAT algorithms scale much better on the SAT-encodings than planing algorithms

[1] For the purposes of the paper, we let the specification, S, be in a dedicated form $S = \mathit{Init} \wedge TR$, where Init is a PLTL formula that represents the initial states and TR is the representation of all the transitions allowed, see §3 for details.

on the original graph formulation. Following this success, SAT solvers have been also used in *Bounded Model Checking* (BMC). In both frameworks, a propositional formula is used to encode the input problem. Planning problems deal with finite paths along a finite graph, hence the encoding is complete w.r.t. the original problem. In model checking, paths within the transition systems are, in general, infinite. SAT-based BMC utilises the encoding of the model-checking problems in satisfiability checking, more precisely, the propositional encoding of statements expressing that there exists a length-k path (along the transition system) that does not satisfy a given property. The BMC increases k until either the SAT's answer is "yes" (i.e. a counter-example is found), or the search becomes intractable, or k reaches a certain bound. The original SAT-based BMC algorithm [3], although complete for finite state, is limited in practice to falsification. Many additional strategies have been introduced to make BCM complete, see [4] for a good survey. There is also a large amount of work, starting with [32], on using SAT solving for improving the satisfiability test of the full PLTL. Recent papers [24, 25] use SAT solvers to seek for a model for an input formula. This model is essentially a graph/automaton produced by the first pass of a two-pass tableau method, which should be followed for testing the fulfilment of eventualities. SAT solvers are called for the generation of all (different) successors of every state in the graph. The authors use well-known temporal equivalences (like $p\mathcal{U}q \equiv (q \vee (p \wedge \circ(p\mathcal{U}q)))$ to compute the successors of the given state. Here, the method utilises the renaming of subformulae containing temporal modalities (such as $p\mathcal{U}q$) by fresh propositional variables and utilise the SAT solving to calculate different successor states of each state in the transition system. This prevents the repeated generation of "propositionally equivalent" states. The relevant heuristics for pruning the search-space and on-the-fly mechanism for testing eventuality fulfilment are introduced in the implementation.

Our proposal uses SAT solvers in a similar way, but is very different in the primary goals. In our method, SAT solvers are employed to calculate a set of the "next" states in the tableau: being in the "current state", SAT solver calculates those successor states in the transition systems that satisfy the negation of the tested property. For that, we do not rename all temporal modalities in the label of the tableau node, but only those of the form $\circ\varphi$, where φ only contains classical operators. Moreover, our proposal of using the one-pass tableau technique (helped, for efficiency, by the SAT solver) is complete for deciding (unbounded) model checking problems and works on infinite traces. The most remarkable difference of our proposal (regarding [24, 25]) is related to the fact that we pursue CMC, i.e. we would like to generate proofs in a calculus for PLTL, hence we use the one-pass context-based tableau along with its dual sequent calculus. In addition, the context-based tableau is particularly well suited for dealing with the specifications of transition systems ("always"-formulae); the context plays the role "of forcing eventualities to be fulfilled as soon as possible" and acts as a semantic constraint that prevents the generation of several states (which are generated in the two-pass approach).

In building the proof, we invoke Isabelle/HOL [30] to verify the construction of the tableaux in order to avoid possible errors caused by the implementation (iv). This external validation is carried out by a sequent calculus TTC which is formalized in Isabelle/HOL and is dual to the one-pass tableau method. We also note that the same reasoning mechanism based on the one-pass tableau is applied in our approach to counter-examples and proofs which makes the tool easy to understand. This facilitates the industrial spreading of CMC.

The remaining of the paper is organised as follows. In §2 we review the technique to construct a one-pass tableau $\tau^{\uparrow}_{pltl}$. In §3 we present one-pass tableau based model checking. This follows by the description of the certification utilising Isabelle in §4. Finally, in §5 we summarise the results and provide an account of future work.

2 One-pass Context-based Tableau

To make the paper self-contained and easier to read, in this section we first recall the syntax and semantics of the underlying logic, PLTL and then we will review the one-pass tableau method.

2.1 Syntax and Semantics of PLTL

▶ **Definition 1** (PLTL Language). *The language of PLTL comprises*

- *A set, Prop, of propositional symbols.*
- *Propositional connectives* $\neg, \wedge, \vee$*, and constants* $\mathbf{T}$ *and* $\mathbf{F}$
- *Future-time temporal connectives, "*$\Box$*" (always), "*$\Diamond$*" (eventually) "*$\circ$*" (at the next moment in time), "*$\mathcal{U}$*" (until), and "*$\mathcal{R}$*" (release).*

▶ **Definition 2** ($\mathsf{WFF}_{\mathsf{PLTL}}$). *The set of well-formed formulae of PLTL, denoted by* WFF_{PLTL}*, is inductively defined as the smallest set satisfying the following.*

- *Any element of Prop,* $\mathbf{T}$ *and* $\mathbf{F}$ *are in* WFF_{PLTL}*.*
- *If A and B are in* WFF_{PLTL} *then so are* $\neg A, A \wedge B, A \vee B, \Box A, \Diamond A, A\mathcal{U}B$*, and* $A\mathcal{R}B$*.*

A literal is either a propositional symbol (a positive literal) or the negation of a propositional symbol (a negative literal).

Definition 1 introduces PLTL with the set of temporal operators that are convenient for our representation in the paper. We note that this set is richer than $\{\mathcal{U}, \circ\}$ which is known to be sufficient to represent all other linear-time temporal operators. For example, '$\Diamond\varphi$' can be defined as $\mathbf{T}\mathcal{U}\varphi$ while $\varphi\mathcal{R}\psi$ can be defined via $\mathcal{U}$ as $\neg(\neg\varphi\mathcal{U}\neg\psi)$ (here and in the remaining of the paper φ and ψ are meta-symbols denoting PLTL formulae).

Formulae of the type $\Diamond\varphi$ and $\varphi\mathcal{U}\psi$ are called *eventualities*, and formulae of the type $\Box\varphi$ are called *always-formulae.*

A model, $\mathcal{M} = s_0, s_1, s_2, s_3, \ldots$, for PLTL formulae is a discrete, linear sequence of states, isomorphic to natural numbers, $\mathbb{N}$. Each state, $s_i, 0 \leq i$, is a set of positive literals, which are satisfied at the i-th moment of time. We write $\langle \mathcal{M}, i\rangle \models \varphi$ to indicate that φ is true in the model $\mathcal{M}$ at the (state) index $i \in N$.

Below we inductively define the relation $\models$ which evaluates PLTL formulae in a model M at i-th moment of time. Note that here we follow so called "anchored" version of PLTL that defines the PLTL validity and satisfiability (see below) at the "beginning" of time, the initial state, s_0 of a model.

- $\langle \mathcal{M}, i\rangle \models \mathbf{T}$
- $\langle \mathcal{M}, i\rangle \not\models \mathbf{F}$
- $\langle \mathcal{M}, i\rangle \models \neg\varphi$ iff $\langle \mathcal{M}, i\rangle \not\models \varphi$
- $\langle \mathcal{M}, i\rangle \models \varphi \wedge \psi$ iff $\langle \mathcal{M}, i\rangle \models \varphi$ and $\langle \mathcal{M}, i\rangle \models \psi$
- $\langle \mathcal{M}, i\rangle \models \varphi \vee \psi$ iff $\langle \mathcal{M}, i\rangle \models \varphi$ or $\langle \mathcal{M}, i\rangle \models \psi$
- $\langle \mathcal{M}, i\rangle \models \circ\varphi$ iff $\langle \mathcal{M}, i+1\rangle \models \varphi$
- $\langle \mathcal{M}, i\rangle \models \Box\varphi$ iff $\langle \mathcal{M}, j\rangle \models \varphi$ for every $j \geq i$.
- $\langle \mathcal{M}, i\rangle \models \Diamond\varphi$ iff there exists $j \geq i$ such that $\langle \mathcal{M}, j\rangle \models \varphi$.
- $\langle \mathcal{M}, i\rangle \models \varphi\mathcal{U}\psi$ iff there exists $j \geq i$ such that $\langle \mathcal{M}, j\rangle \models \psi$ and for every k, $i \leq k < j$, we have $\langle \mathcal{M}, k\rangle \models \varphi$.
- $\langle \mathcal{M}, i\rangle \models \varphi\mathcal{R}\psi$ iff for every $j \geq i$, either $\langle \mathcal{M}, j\rangle \models \psi$ or there exists k such that $i \leq k < j$ and $\langle \mathcal{M}, k\rangle \models \varphi$.

▶ **Definition 3** (PLTL satisfiability, validity). *If, for a formula φ, there exists a model, M, such that $(M,0) \models \varphi$ then φ is satisfiable. Formula φ is called valid if it is satisfiable in all models.*

In the remaining of the paper we will use capital letters $\Phi, \Psi, \Delta, \ldots$ to denote sets of PLTL formulae. The semantics above can be extended to sets of formulae in the standard way: given a set of PLTL formulae $\Phi = \gamma_1, \gamma_2, \ldots \gamma_n$, the following holds: $\langle \mathcal{M}, i\rangle \models \Phi$ iff $\langle \mathcal{M}, i\rangle \models \gamma_k$, for all $k,\ 1 \leq k \leq n$.

2.2 Useful PLTL properties

In this section we recall those PLTL syntactic and semantic properties that are useful for our tableau construction. First, the tableau procedure will take as an input PLTL formulae converted to their negated normal forms (NNF).

▶ **Definition 4** (Procedure for obtaining NNF). *For a given PLTL formula, φ, push the negations in φ inward until they are applied only to propositions. This involves applying the standard set of rewrite rules used to obtain NNF in classical logic (including $\neg\mathbf{T} \longrightarrow \mathbf{F}$ and $\neg\mathbf{F} \longrightarrow \mathbf{T}$) with the additional transformations for temporal operators:*

$$\neg\bigcirc\varphi \longrightarrow \bigcirc\neg\varphi \qquad \neg\Box\varphi \longrightarrow \Diamond\neg\varphi \qquad \neg\Diamond\varphi \longrightarrow \Box\neg\varphi$$
$$\neg(\varphi\mathcal{U}\psi) \longrightarrow \neg\varphi\mathcal{R}\neg\psi \qquad \neg(\varphi\mathcal{R}\psi) \longrightarrow \neg\varphi\mathcal{U}\neg\psi$$

The following result [26] can be easily established.

▶ **Proposition 5** (Translation into NNF preserves satisfiability). *For any* PLTL *formula φ, the following holds* $\langle \mathcal{M}, 0\rangle \models \varphi$ *iff* $\langle \mathcal{M}, 0\rangle \models \mathit{NNF}(\varphi)$.

As a simple example, $\mathsf{NNF}(\neg\bigcirc\Box\neg a) = \bigcirc\Diamond a$. We will utilise this in §3.

In what follows we deal with sets of formulae in NNF. Literals and formulae in NNF of the form $\mathbf{F}$, $\mathbf{T}$ and $\bigcirc\varphi$ are called *elementary*, the remaining formulae are subsequently called *non-elementary*. In addition, sets of elementary formulae are also called elementary.

▶ **Definition 6** (Consistent set of PLTL formulae in NNF). *A set of PLTL formulae in NNF is consistent if it does neither contain $\mathbf{F}$, nor $\{\varphi, \mathit{NNF}(\neg\varphi)\}$ for any formula φ. Otherwise it is called inconsistent.*

Note that the above notion of consistency is syntactic. To check whether $\{\varphi, \psi\}$ is inconsistent we test if $\varphi = \mathsf{NNF}(\neg\psi)$. The cost of this check is linear on the length of the formula.

Since our transition system specifications are given by sets of PLTL formulae and PLTL has the *finite model property* [33] we can consider its interpretation over *cyclic* structures. Noting that an infinite sequence $s_0, s_1, \ldots, s_k, \ldots$ induces the successor relation, R, such that $(s_i, s_{i+1}) \in R$ for all $i \in \pi$, we define below the notions of a cyclic sequence, cyclic path and cyclic model.

▶ **Definition 7** (Cyclic Sequence, Cyclic Path). *Let π be a finite sequence of states $\pi = s_0, s_1, \ldots, s_j$. Then*

- *π is cyclic iff there exists $s_i,\ 0 \leq i \leq j$ such that $(s_j, s_i) \in R$.*
- *A sequence $s_i, \ldots, s_j$ is a loop with a cycling element s_i abbreviated as $\langle s_i, \ldots, s_j\rangle^\omega$.*
- *A cyclic path over a cyclic sequence π is an infinite sequence $\xi(\pi) = s_0, s_1, \ldots, s_{i-1}\langle s_i, s_{i+1}, \ldots, s_j\rangle^\omega$.*

▶ **Definition 8** (Cyclic Model). *A model $\mathcal{M}$ is cyclic if it is a cyclic path.*

Now, assuming that PLTL formulae are interpreted over cyclic models, we overview the construction of the one-pass tableau $\tau^{\uparrow}_{pltl}$ applied to some input Σ, in symbols $\tau^{\uparrow}_{pltl}(\Sigma)$, adapting the technique developed in [15]. We will see, in the subsequent sections, the benefits of its main feature – checking the validity of the given input in "one pass", without the second "pass" where an auxiliary graph is built to check if all the eventualities are satisfied or not.

▶ **Definition 9** (Tableau, Consistent Node, Closed branch). *A one-pass tableau for a set of formulae Σ, abbreviated as $\tau^{\uparrow}_{pltl}(\Sigma)$ is a labelled tree T, where nodes are labeled with sets of formulae, such that the following two conditions hold:*

(a) *The root is labelled by the tableau input, Σ.*

(b) *Any other node, m, is labelled with sets of formulae as the result of the application of one of the expansion rules to the parent node, n.*

A node $n \in T$ is consistent, abbreviated as $n_{\top}$, if its label is a consistent set of formulae (see Def. 6), else n is inconsistent, abbreviated as $n_{\bot}$.

If a branch, b, of T, contains an inconsistent node $n_{\bot}$, then b is closed, else b is open.

Informal Introduction to one-pass tableau. In the set of expansion rules, on the top of the standard $\alpha - \beta$ rules, we also have β^{+} rules that are characteristic (and crucial!) for our construction. These rules (which were originally introduced in [14, 15]) reflect our dedicated account of the eventualities, namely, we treat an eventuality as occurring in some *context*. By the context of the selected eventuality, we understand a collection of all other formulae within the label of the node. Subsequently, β^{+} rules use the context to force eventualities to be fulfilled as soon as possible. The tableau expansion rules apply repeatedly until they produce an inconsistent node, $n_{\bot}$, or a node with the labels that already occurred within the given path. In the former case the expansion of the given branch terminates with $n_{\bot}$ as a leaf. In the latter case, a repetitive node in the branch witnesses that the branch is open. Once no more expansion ($\alpha - \beta, \beta^{+}$ type) rules are applicable to the given branch with the last consistent node $n_{\top}$, the expansion rules ensure that its labelling is similar to a "state" in the standard temporal tableau. Then the "next-state" rule applies which generates successors with the labels that are arguments of all $\circ$ modalities and the whole cycle of applying the expansion and the "next-state" rules is repeated until the tableau construction terminates. The nature of our rules ensures that the terminated tableau is either closed, indicating that the input does not have a model, hence unsatisfiable, or open, indicating a model for the tableau input.

2.3 $\tau^{\uparrow}_{pltl}$ Rules

Recall that all formulae in the input of the tableau have been already transformed into their NNF. Presenting α-, β- and "next-state" rules for the construction of the semantic tableaux, we assume that these apply respectively, to α-formulae, β-formulae and "next"-formulae such that α_1 denotes the set of formulae in the conclusion of an α-rule, while β_1, β_2 denote the sets of formulae in the alternative conclusions of a β-rule, and γ_1 denotes the result of "jumping" from a state to a pre-state.

The sets of $\alpha - \beta$ rules are given in Table 1. These are standard in temporal tableaux construction (see, for instance, [2]).

Table 1 $\alpha - \beta$-rules.

	α	α_1
$(\wedge)$	$\varphi \wedge \psi$	φ, ψ
$(\Box)$	$\Box\varphi$	$\varphi, \circ\Box\varphi$

	β	β_1	β_2
$(\mathcal{U})$	$\varphi\mathcal{U}\psi$	ψ	$\varphi, \circ(\varphi\mathcal{U}\psi)$
$(\mathcal{R})$	$\varphi\mathcal{R}\psi$	φ, ψ	$\psi, \circ(\varphi\mathcal{R}\psi)$
$(\vee)$	$\varphi \vee \psi$	φ	ψ
$(\Diamond)$	$\Diamond\varphi$	φ	$\circ\Diamond\varphi$

Table 2 β^+-rules, where

- Δ is a (possibly empty) set (conjunction) of formulae,
- If $\Delta \neq \emptyset$ then Δ' is a set (conjunction) of all elements of Δ except for $\Box$ -formulae and $\neg\Delta'$ is the disjunction of all negated elements of Δ', else $\neg\Delta'$ is $\mathbf{F}$.

	β	β_1	β_2
$(\mathcal{U})^+$	$\Delta, \varphi\mathcal{U}\psi$	Δ, ψ	$\Delta, \varphi, \circ((\varphi \wedge (\mathsf{NNF}(\neg\Delta')))\mathcal{U}\psi)$
$(\Diamond)^+$	$\Delta, \Diamond\varphi$	Δ, φ	$\Delta, \circ((\mathsf{NNF}(\neg\Delta'))\mathcal{U}\varphi)$

β^+ rules (see Table 2) are crucial in the construction of $\tau^{\uparrow}_{pltl}$ as they use the so-called *context*, Δ, to force the eventuality $\varphi\mathcal{U}\psi$ to be fulfilled as soon as possible. We illustrate this concept on the $(\mathcal{U})^+$ rule. When $(\mathcal{U})^+$ is applied to a node labelled by a set of formulae $\{\Delta, \varphi\mathcal{U}\psi\}$, then the context is Δ and the resulting labelling of the next node contains the formula $(\varphi \wedge \neg\Delta)\mathcal{U}\psi$, where $\neg\Delta$ means the disjunction of all negated elements of Δ. Therefore, if ψ is not satisfied, then $\neg\Delta$ also belongs to the label of that node. This means that the context, Δ, of the previous label is not repeated. As Δ is a finite set/conjunction of formulae and $\neg\Delta$ is the finite disjunction of the negations of the formulae in Δ, the $(\mathcal{U})^+$ rule forces at least one formula in Δ to be falsified during the transition from the previous node to the subsequent one, whenever ψ is not satisfied. Note that the $(\mathcal{U})^+$ rule only applies to some selected eventuality, and in this sense, the unique eventuality, which becomes marked. Each application of the $(\mathcal{U})^+$ rule to the set $\{\Delta, \varphi\mathcal{U}\psi\}$ introduces the so-called *next-step variant* $(\varphi \wedge (\mathsf{NNF}(\neg\Delta'))\mathcal{U}\psi$ where Δ', as we know, is a conjunction of all elements of Δ except for $\Box$ -formulae, which keeps the mark for the selected eventuality. Note that any formula of the type $\Box\varphi$, which was a member of Δ, will be repeated forever and if we keep it in Δ' it would appear in the resulting $\mathsf{NNF}(\neg\Delta')$ as a disjunct $\Diamond\neg\varphi$ which would never be satisfied. Hence, any $\Box$- formula can be immediately dropped when we form Δ'. For the soundness of the construction, each node of the tableau must have at most one marked eventuality, i.e. the one to which the $(\mathcal{U})^+$ has been applied. Note that when a node of the tableau does not contain any marked eventuality, then one of them is randomly marked.

The $(\mathcal{U})^+$ rule allows us to avoid the construction of the auxiliary graph of stages (the second pass in two-pass tableau methods) which is used to determine whether all eventualities are satisfied or not.

The other β^+ rule, $(\Diamond)^+$, is obviously derivable from the $(\mathcal{U})^+$ rule. However, we present it as part of the tableau as its application makes proofs more transparent to the reader and the user of the tableau as a model checker. We will explain this in the subsequent sections of the paper. It is worth noting that, because in the β^+ rules, $\neg\Delta' = \mathbf{F}$ whenever $\Delta = \emptyset$, the β rules $(\mathcal{U})$ and $(\Diamond)$ become particular cases of β^+ rules $(\mathcal{U})^+$ and $(\Diamond)^+$ when $\Delta = \emptyset$. In this case the β_2 child of the $(\mathcal{U})^+$ rule is reduced to $\varphi, \circ(\mathbf{F}\mathcal{U}\psi)$, from which we can derive the β_2 child of the $(\mathcal{U})$ rule.

Table 3 The next-state rule (Σ_1 is a set of literals).

	γ	γ_1
$(\circ)$	$\Sigma_1, \circ(\Sigma_2)$	Σ_2

For the formulation of the next-state rule $(\circ)$, we first introduce the following notation. Given a set of formulae, Φ, we denote by $\circ(\Phi)$ the set $\{\circ\varphi \mid \varphi \in \Phi\}$. The next-state rule (see Table 3), is applied to jump from a node labelled by $\Sigma_1, \circ(\Sigma_2)$ to a new state, which is labeled by the set Σ_2

3 One-pass tableau based Model Checking

In this section we explain how the model checking can be performed using the one-pass tableau method described in §2. First, we define the procedure to follow, then we explain the optimization we pursue by embedding a (propositional) solver; finally, we present an example, showing how beneficial it can be for the users of CMC to obtain an explicit and understandable formal proof.

Our model checker receives as its input a specification, S, of the given transition system, and the given property P, both written in the PLTL language. Whereas P is any PLTL formula, transition system specifications are restricted to a sublanguage. Our specification language is inspired in the so-called *constraint style* specification language used in the well-known model checker NuSMV (introduced in [8]). The specification S consists of a set *Init* that is a non-temporal (or classical) formula, and a conjunction (set) TR of formulae of the form $\Box\rho$ where ρ is a boolean combination of literals and $\circ\ell$ where ℓ is a literal. $S = \mathit{Init} \wedge TR$ is the system specification such that

- A state is initial if and only if it satisfies the formula *Init*, and
- Any pair "(current state, next state)" is in the transition relation if, and only if, it satisfies TR.

In Example 10 we provide the *Init* and TR formulae that represent a specific transition system. Then, given a specification S and a property P, the model checker decides whether any model of S satisfies P, by deciding if $S \cup \{\neg P\}$ is unsatisfiable or not. For that, S and $\neg P$ are firstly converted into NNF. The tableau method $\tau^{\uparrow}_{pltl}$ is suitable for deciding the (un)satisfiability problem $\mathsf{NNF}(S \cup \{\neg P\})$. It explicitly tries to generate a model of $\mathsf{NNF}(S \cup \{\neg P\})$. The results are interpreted as follows:

- If a cycle is found in a tableau branch, this branch is open, and represents a model that satisfies the set of formulae with which the tableau has been called. Consequently, this is a counter-model proving that the system S does not satisfy the property P.
- If the tableaux closes, i.e all its leaves are inconsistent sets, it means that the tableaux input is unsatisfiable, hence, all models of S satisfy the property P.

This way of performing model checking is a particular case of the method described in Section 2. It brings us the following benefits: first of all, the tableau is built on-the-fly, allowing structures not to deal with eventualities fulfillment; due to the use of the context, the eventualities are satisfied as soon as possible. In terms of the implementation, all branches are completely independent so they can be parallelised without shared data. Finally, we have a potential memory improvement by only requiring to keep traces (of the branch that we deal at the moment).

On the other hand, a large proportion of the computational effort is spent on classical propositional reasoning. Since the specification, S, of the system is the most determining factor, this is especially inefficient when the specified system is large. However, S involves very

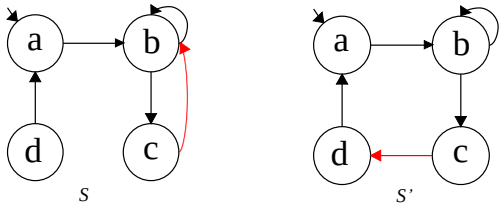

Figure 3 Example showing how the approach can help in detecting errors in system specification.

simple temporal formulae: always-formulae with arguments that are Boolean combinations of literals and literals preceded by "next". TR is a conjunction of the formulae of the form $\Box\rho$, and according to the tableau rules (and semantics), ρ and $\Box\rho$ are maintained in all states. Therefore, renaming in ρ the formulae $\circ\ell$ by fresh literals ℓ', we make ρ purely propositional. Hence, for a more efficient implementation of $\tau^{\uparrow}_{pltl}$, we propose to add a SAT solver to carry out the propositional reasoning in the tableau. Initially, we pass to the solver *Init* and the renamed formulae extracted from TR. Then, the tableau rules work on the temporal formulae (taken from the NNF of the negated property). Subsequently, in every node which is a "state" (i.e. a node labelled by an elementary set of formulae) the SAT solver is called adding to its input all formulae that appear new in the node and are either non-temporal (classical) formulae, or formulae of the type of the TR ones, but adequately translated. The SAT solver returns propositional models (atoms and atoms with prime apostrophe) defining all possible transitions to the next state that do not contradict (yet) $\neg P$. To get each of the next states, the variables are renamed back from ℓ' to $\circ\ell$ and the next-state rule is applied.

▶ **Example 10.** Let us introduce a running example (Figure 3), to illustrate how the tableau method works and the role of the solver, given the specification written in the form $Init \wedge TR$. The *Init* formula of both transition systems in Figure 3 is: $a \wedge \neg b \wedge \neg c \wedge \neg d$. Let us suppose that the transition system S' on the right-hand side of Figure 3 is the system we intend to specify, but a misprint in the system specification produces a wrong specification S in the left-hand side of Figure 3. That is, we mistakenly specify that there is a transition "from c to b" instead of the intended one: "from c to d". The resulting specification is:

$$\begin{aligned} TR = \{\ & \Box\,(a \rightarrow (\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)),\\ & \Box\,(b \rightarrow ((\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)\ \vee\ (\circ\neg a \wedge \circ\neg b \wedge \circ c \wedge \circ\neg d))),\\ & \Box\,(c \rightarrow (\circ\neg a \wedge \circ b \wedge \circ\neg c \wedge \circ\neg d)),\\ & \Box\,(d \rightarrow (\circ a \wedge \circ\neg b \wedge \circ\neg c \wedge \circ\neg d))\ \} \end{aligned}$$

Now, suppose that we want to check if S satisfies the PLTL formula $\circ\Box\neg a$. Converting its negation to NNF we get the formula $\circ\Diamond a$. Then, we call $\tau^{\uparrow}_{pltl}$ with the following input – the label of the initial node – $TR \cup \{Init, \circ\Diamond a\}$. Thinking on the system S', we expect to get a sequence $\langle a, b, c, d, a\rangle$ as a counter-example, since this sequence is a model of $\circ\Diamond a$ and the correct system S'. However, $S \cup \{\circ\Diamond a\}$ is unsatisfiable and our model checker generates a closed tableau for it.

In Figure 4 we depict a big-step version of that closed tableau. This one represents the different branches of the tableaux enabling an easy follow-up of the runs across the system S. In the rest of this section and in Section 4 we discuss the utility of the closed tableau and its dual sequent proof to find an error when defining S instead of the intended S'. The tableau root, in Figure 4, contains the $Init = a \wedge \neg b \wedge \neg c \wedge \neg d$, the negated property, $\circ\Diamond a$, and TR which represents the system's transitions. The tableau method applies its rules until a state

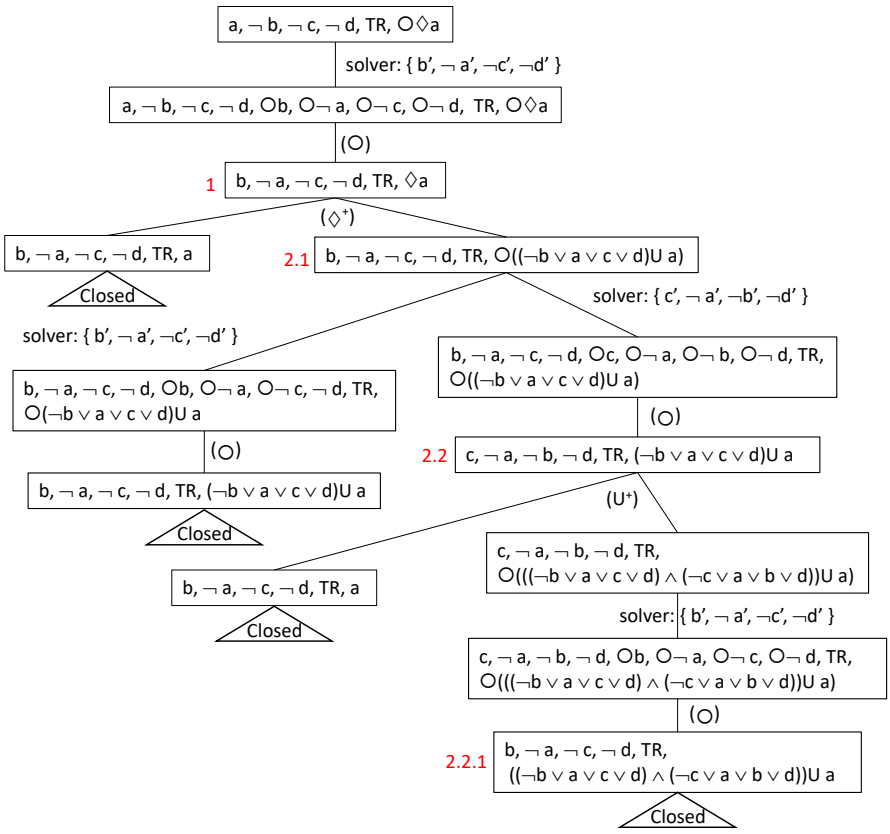

Figure 4 A big-step representation of the closed tableau for $S \cup \{\circ\Diamond a\}$.

is reached. It is now, when the solver is called to work with the propositional translation of TR and the remaining propositional formulae in the current node. This procedure gives us, one by one, the different models that satisfy the (translated) specification TR, and the current state (propositional formulae with which it is called). These models, (built for the formulae - results of the renaming of each "next" formula $\circ l$ by ℓ'), are transformed back to PLTL formulae by re-instantiating the renamed $\circ\ell$. Then, the next-state rule is applied and the tableaux continues working applying the temporal rules to the non-elementary temporal formulae (that come from the negated property). When the tableau is closed, a different model is supplied by the solver, if there is any. For example, for node 2.1 of the tableau, two different propositional models are supplied. First, the solver returns a model in which only b' (that is, $\circ b$) is true. It applies the next rule and all the subsequent branches close. Returning to 2.1., the solver supplies a new propositional model in which it is now c' (i.e. $\circ c$) that is true. In short, the solver is in charge of returning in each state all possible models of the next state (if any). In our running example, if users analyse the tableau they can see the reason for the property not to be fulfilled: the trace $\langle root, 2.1, 2.2, 2.2.1\rangle$ describes the transitions (omitting the negated literals) $a \Rightarrow b \Rightarrow c \Rightarrow b$. By comparing that trace with the intended specification (S') the user will be able to find an error.

4 Isabelle Proofs as User-checkable Certificates

We have codified a version of the calculus TTC in [15] (for formulae in NNF) as an Isabelle/HOL theory ([30]). The file `TTC_Calculus.thy` contains the encoding of the sequent rules that we will explain in this section, and another file `TTC_Soundness.thy` provides the soundness proof of them. On the top of this theory we define (for the running example) a transition system, S, as a collection, TR, of named formulae (in the fixed syntax). Thus, we introduce the (binary) transition relation between states and the formula *Init* specifying the initial states of the system. We automatically prove lemmas asserting that falsity $\mathbf{F}$ is derivable from the union of S and the negation (in NNF) of a fixed property φ. Such a proof object guarantees that φ is satisfied in every run of S, and can be independently and efficiently verified by the interactive theorem-prover Isabelle/HOL, providing a machine-checked certificate. Moreover, when the proof is unexpected, interactive theorem-provers enable to use this certificate to analyze the transition system seeking for specification errors. For that, we automatically generate an Isabelle proof that allows the user to check the successive subgoals of the proof. This Isabelle proof can be generated with different levels of granularity to facilitate different levels of analysis. In this section, we explain the main ingredients of this Isabelle/HOL development. The corresponding files can be download from `http://github.com/alexlesaka/OnePassTableau`.

We use a datatype PLTL_formula to define the syntax of the considered formulae, which are the two boolean constants ($\mathbf{T}$ and $\mathbf{F}$), the atoms (strings preceded by constructor Var, or shortly V), classical connectives (preceded by a dot, to avoid conflicts) of negation ($.\neg$), conjunction ($.\wedge$), disjunction ($.\vee$), and implication ($.\longrightarrow$), along with temporal connectives for next, until, release, eventually and always. For automating the Isabelle proof, we add two extra connectives $\ddot{\mathcal{U}}$ (the until operator surrounded by dieresis) to denote the selected eventuality in goals and $\ddot{\circ}$ to mark the sequents formed by elementary formulae. We define the TTC rules by an inductive binary relation (predicate) TTC_proves, which is denoted "$\vdash$" in infix notation. The first argument of $\vdash$ is a set of PLTL formulae (implemented as an ordered list without repeated elements) and the second is a PLTL formula. By the construction of the calculus, the second argument is always the constant $\mathbf{F}$, but (for clarity) we prefer to keep $\vdash$ as a binary relation, and to explicitly represent that falsehood in the right-hand side of each goal. The Isabelle definition of $\vdash$ includes the two contradiction rules:

$$\begin{array}{ll} \mathit{TTC\_Ctd1}: & \varphi \mathbin{.\in} \Delta \Longrightarrow (\mathsf{NNF\_Neg}\ \varphi) \mathbin{.\in} \Delta \Longrightarrow \Delta \vdash \mathbf{F} \\ \mathit{TTC\_Ctd2}: & \mathbf{F} \mathbin{.\in} \Delta \Longrightarrow \Delta \vdash \mathbf{F} \end{array}$$

where $.\in$ is the user-defined infix operator for member of a list, and the user-defined function NNF_Neg computes the negation normal form of the negation of a given formula, i.e. $(\mathsf{NNF\_Neg}\ \varphi) = \mathsf{NNF}(.\neg\varphi)$. We also encode the traditional rules for classical and temporal connectives:

$$\begin{array}{ll} \mathit{TTC\_T}: & \Delta \vdash \mathbf{F} \Longrightarrow \mathbf{T}\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_And}: & \varphi \bullet \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi . \wedge \psi)\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_Or}: & \varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi . \vee \psi)\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_Imp}: & (\mathsf{NNF\_Neg}\ \varphi) \bullet \Delta \vdash \mathbf{F} \Longrightarrow \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi . \longrightarrow \psi)\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_Alw}: & \varphi \bullet \circ\Box\varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\Box\varphi)\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_R}: & \varphi \bullet \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \psi \bullet \circ(\varphi \mathcal{R} \psi) \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathcal{R} \psi)\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_Evt}: & \varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \circ\Diamond\varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\Diamond\varphi)\ \#\ \Delta \vdash \mathbf{F} \\ \mathit{TTC\_U}: & \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \varphi \bullet \circ(\varphi \mathcal{U} \psi) \bullet \Delta \vdash \mathbf{F} \Longrightarrow (\varphi \mathcal{U} \psi)\ \#\ \Delta \vdash \mathbf{F} \end{array}$$

where # is the standard *cons* constructor of lists and • is our user-defined operator for insert an element φ in the correct position of an ordered list Δ (if φ is already in Δ, then the result is Δ itself). For automating proofs, we have defined an order on the set of PLTL formulae where the minimal formulae are literals, and formulae with connectives are lexicographic ordered according to the following order on the set of connectives are ordered (from lowest to highest) as follows: $\ddot{}\circ\ddot{}$, $\circ$, $.\wedge$, $.\vee$, $.\longrightarrow$, $\Box$, $\mathcal{R}$, $\Diamond$, $\mathcal{U}$, $\ddot{}\mathcal{U}\ddot{}$. We order the antecedent of the sequent, in decreasing order, at the beginning of any proof using the following extra rule:

$$TTC\_Interchange: \quad (\mathsf{sort}\ \Delta) \vdash \mathbf{F} \Longrightarrow \Delta \vdash \mathbf{F}$$

Then, every application of a rule preserves the order in the generated subgoals by means of the operator • that inserts each formula in the correct place. As a consequence, the first formula of any sequent is a *non-elementary* formula (see Section 2), if there exists at least one. Moreover, the first formula is the eventuality, if there exists at least one, and it is the selected eventuality whenever the selection has already been done. The rules applied to the selected eventuality are:

$$\begin{array}{ll}
TTC_Evt_Plus: & \varphi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \circ((\mathsf{negCtxt}\ \Delta)\mathcal{U}\varphi) \bullet \Delta \vdash \mathbf{F} \\
& \Longrightarrow \Diamond\varphi\ \#\ \Delta \vdash \mathbf{F} \\
TTC_U_Plus: & \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \varphi \bullet \circ((\varphi\ .\sqcap.\ (\mathsf{negCtxt}\ \Delta))\ddot{}\mathcal{U}\ddot{}\psi) \bullet \Delta \vdash \mathbf{F} \\
& \Longrightarrow \varphi\mathcal{U}\psi\ \#\ \Delta \vdash \mathbf{F} \\
TTC_U_Sel: & \psi \bullet \Delta \vdash \mathbf{F} \Longrightarrow \varphi \bullet \circ((\varphi\ .\sqcap.\ (\mathsf{negCtxt}\ \Delta))\ddot{}\mathcal{U}\ddot{}\psi) \bullet \Delta \vdash \mathbf{F} \\
& \Longrightarrow \varphi\ddot{}\mathcal{U}\ddot{}\psi\ \#\ \Delta \vdash \mathbf{F}
\end{array}$$

where $(\mathsf{negCtxt}\ \Delta)$ is the negation of the context Δ, that is a disjunction of the negations (in NNF) of all formulae in Δ excepting the formulae of the form $\Box\varphi$. In addition, the operator $.\sqcap.$ is a conjunction up to subsumption, hence we avoid adding subsumed disjunctions, in particular, adding duplicated disjunctions. Note that the premises of *TTC_U_Sel* are really a copy of the premises of *TTC_U_Plus*.Consequently, *TTC_U_Plus* is applied at the first time when the eventuality has been just selected, whereas *TTC_U_Sel* is applied after that, while it is kept selected. When all formulae in the sequent are elementary we apply the following rule:

$$TTC\_Next\_State: \quad (\mathsf{next\_state}\ \Delta) \vdash \mathbf{F} \Longrightarrow \Delta \vdash \mathbf{F}$$

This rule applies when all the formulae in Δ are elementary (see Section 2) and the function next_state has filtered the formulae in Δ starting by the $\circ$ operator removing from them this operator.

The transition system in S on the left-hand of Figure 3 is defined as the list $TR = [T1,\ T2,\ T3,\ T4]$ of PLTL formulae:

$$\begin{array}{lcl}
T1 & = & \Box((V\ a)\ .\longrightarrow\ (\ \circ(V\ b)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ ) \\
T2 & = & \Box((V\ b)\ .\longrightarrow\ (\ (\circ(V\ b)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ .\vee \\
 & & \qquad (\circ(V\ c)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ b))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ ) \\
T3 & = & \Box((V\ c)\ .\longrightarrow\ (\ \circ(V\ b)\ .\wedge\ \circ(\ .\neg\ (V\ a))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\ .\neg\ (V\ d)))\ ) \\
T4 & = & \Box((V\ d)\ .\longrightarrow\ (\ \circ(V\ a)\ .\wedge\ \circ(\ .\neg\ (V\ b))\ .\wedge\ \circ(\ .\neg\ (V\ c))\ .\wedge\ \circ(\neg\ (V\ d)))\ )
\end{array}$$

along with $Init = (V\ a)\ .\wedge\ .\neg\ (V\ b)\ .\wedge\ .\neg\ (V\ c)\ .\wedge\ .\neg\ (V\ d)$. Note that in the transition specifications the "next" operator is completely distributed over conjunction and disjunction. Hence, contradictions of the form $\circ(V\ x)$, $\circ.\neg\ (V\ x)$ are detected. Otherwise, the contradiction $V\ x$, $\neg\ (V\ x)$ should be detected at the next state.

We have implemented, with the help of the Eisbach tools [27], two prototypes of automatic solvers: one big-step and one small-step. These two solvers print into a text file the "apply" instructions of the Isabelle proof, at the same time that they prove the lemma. The proof of lemma runningExample_bigStep_proof is given in Figure 5. It proves the property $S \ @ \ [\circ\Diamond(V\ a)] \vdash \mathbf{F}$ by a list of "apply" instructions. This proof can be found in file `ProofGeneration.thy` and it is a big-step proof that enables the user to check the transitions of the system S that are performed when checking whether it satisfies a property φ, that is checking the unsatisfiability of $S \cup \{\neg\varphi\}$, in fact the derivability of the sequent $S, \neg\varphi \vdash \mathbf{F}$.

```
lemma runningExample_bigStep_proof: "S @ [○(◇(V ''a''))] ⊢ F"
apply (rule TTC_Interchange, simp add: S_def TR_def T1_def T2_def T3_def T4_def Init_def) (*1*)
apply one_step_solver                                                                    (*2*)
apply (all ‹rule TTC_Next_State›; simp)                                                  (*3*)
apply (fold T1_def T2_def T3_def T4_def)                                                 (*4*)
apply (simp add: T1_def T2_def T3_def T4_def Init_def)                                   (*5*)
apply one_step_solver                                                                    (*6*)
apply (all ‹rule TTC_Next_State›; simp)                                                  (*7*)
apply (fold T1_def T2_def T3_def T4_def)                                                 (*8*)
apply (simp add: T1_def T2_def T3_def T4_def Init_def)                                   (*9*)
apply one_step_solver                                                                    (*10*)
apply (all ‹rule TTC_Next_State›; simp)                                                  (*11*)
apply (simp add: T1_def T2_def T3_def T4_def Init_def)                                   (*12*)
apply one_step_solver                                                                    (*13*)
apply (all ‹rule TTC_Next_State›; simp)                                                  (*14*)
apply (fold T1_def T2_def T3_def T4_def)                                                 (*15*)
apply (simp add: T1_def T2_def T3_def T4_def Init_def)                                   (*16*)
apply one_step_solver                                                                    (*17*)
  done
```

Figure 5 The big-step lemma proof.

The (proof) method one_step_solver systematically applies the TTC_Calculus rules until we obtain a set of non-proved subgoals with antecedents exclusively formed by the elementary formulae. Hence, the rule TTC_Next_State is applied to all subgoals, which depict (as "Proof state") all subgoals related to a possible next state of the system, that is, after all possible transitions from the current state. For clarity, we fold the transition relations to their names. Hence, after the "apply" in line 3 (Figure 5), the user can see that there is only one subgoal:

$$[\Diamond(V\ a),\ T4,\ T2,\ T3,\ T1,\ .\neg(V\ d),\ .\neg(V\ c),\ .\neg(V\ a),\ (V\ b),\ ] \vdash \mathbf{F}$$

This means the only state that is reachable from the initial one is the state that satisfies b. That corresponds to the node marked with 1 in Figure 4. After the "apply", line 7, there are two subgoals that correspond, respectively, to the nodes marked with 2.1 and 2.2 in Figure 4. Thus, from the state, which satisfies exactly b, the system can reach either the same state again or the state satisfying exactly c. In both cases, the property to check is $(\neg b \vee a \vee c \vee d)\ \mathcal{U}\ a$. The goal 2.1 is proved, whereas the "apply", line 11, (Figure 5) generates the subgoal corresponding to node 2.2 in Figure 4. This subgoal corresponds to the transition from the state that satisfies b to the state that satisfies c. The property to check at this state is $(\neg b \vee a \vee c \vee d)\ \mathcal{U}\ a$. After the "apply" in line 14, the subgoal which corresponds to the node 2.2.1 in 4 is reached; here the property to check is $((\neg b \vee a \vee c \vee d) \wedge (\neg c \vee a \vee b \vee d))\ \mathcal{U}\ a$. The proof of this subgoal completes the proof of the lemma, that has explored the two possible runs $\langle a, b, b\rangle$ and $\langle a, b, c, b\rangle$ where the property $\circ\Diamond a$ is not satisfied. This shows the inability of the transition system to reach the state that satisfies d, which reveals an error in the specification that makes unreachable the state that satisfies a.

The "apply(fold ...)" instructions in Figure 5 are only to show, in subgoals, the names ($T1$, $T2$, $T3$, and $T4$) instead of the corresponding PLTL formulae defining the transitions, for brevity and clarity. After that, we must use "apply(simp add: ...)" to enable the application of the rules.

We have also implemented a method one_step_solver_print which prints into a file of text all applications of the TTC rules that are hidden in the big-step proof. Indeed, it is a printing version of the method one_step_solver. Calling one_step_solver_print, instead of one_step_solver, we can generate a small-step proof (see lemma runningExample_small-Step_proof) for the user wishing to check the Isabelle's subgoals step by step. This proof is the result of the substitution, in the big-step proof, of each of the five occurrences of apply one_step_solver by the list of "apply" instructions of TTC_Calculus rules. The method one_step_solver_print prints such a list, in a text file, while it is solving the goal.

5 Conclusions

In this paper we have presented a novel framework of applying a Certified Model Checking methodology. Our method involves the representation of the system specification, S, in a specific format that reflects the dynamic behaviour of the system. First, it involves the formalization of the "initial conditions", (the *Init* PLTL formula), that specifies the initial states of the model to be build. Second, we use the representation, TR, of the transition relation to build a state space of the system, as a "global invariant". Finally, the property, P, to be checked against the specification S is written as a PLTL formula. This task is performed by a one-pass temporal tableau, $\tau^{\uparrow}_{pltl}$. It takes $S, \neg P$ as its input (converted into the NNF). For an eventuality to be fulfilled, the tableau technique essentially uses its context. Tableau rules that deal with the eventualities in NNF($\neg P$), force their fulfilment "as soon as possible". This process is optimised by a SAT solver, which tackles non-temporal content of the tableau nodes. The tableau method, in one pass, either returns a negative answer, producing a counter-example, thus showing that P is not satisfied by S, or verifies P against the system specification. In the latter case, our method generates an explicit and easily readable evidence in Isabelle/HOL. In this way the tableau result is formally proven and the user can review the test to make sure that everything is working as expected (or that no errors have been made with the specification).

Our future work will cover three directions. First, it is further work on the implementation of and experimentation with the one-pass tableau and the embedded SAT solver. Second, the developed Isabelle automatic solver for proof generation is only an initial prototype and we will improve its efficiency. Finally, note that the one-pass tableau method has been also developed for the branching-time setting, tackling Computation Tree Logic CTL ([6]), widely used in model checking, and for a richer logic, $ECTL^{\sharp}$ ([5]). This will enable us to extend the use of the one-pass tableau as a model checker to the branching-time setting, as the extensions are conceptually intuitive.

References

1 Hasan Amjad. Programming a Symbolic Model Checker in a Fully Expansive Theorem Prover. In *Theorem Proving in Higher Order Logics*, pages 171–187. Springer Berlin Heidelberg, 2003. doi:10.1007/10930755_11.

2 Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer-Verlag, London, 2012. doi:10.1007/978-1-4471-4129-7.

3 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic Model Checking Using SAT Procedures Instead of BDDs. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 317–320, New York, NY, USA, 1999. ACM. doi:10.1145/309847.309942.

4 Armin Biere and Daniel Kröning. SAT-based model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 277–303. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-10575-8_10.

5 Alexander Bolotov, Montserrat Hermo, and Paqui Lucio. Extending Fairness Expressibility of ECTL+: A Tree-Style One-Pass Tableau Approach. In Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek, editors, *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TIME.2018.5.

6 Kai Brünnler and Martin Lange. Cut-free sequent systems for temporal logic. *The Journal of Logic and Algebraic Programming*, 76(2):216–225, 2008. doi:10.1016/j.jlap.2008.02.004.

7 CENELEC and EN50128. 50128. *Railway applications-Communication, Signaling and Processing Systems-Software for Railway Control and Protection Systems*, 2011.

8 Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, March 2000. doi:10.1007/s100090050046.

9 Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In Gianpiero Cabodi and Satnam Singh, editors, *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK*, pages 52–59. IEEE, 2012.

10 Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, pages 52–71, 1981. doi:10.1007/BFb0025774.

11 Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

12 Marco Comini, Laura Titolo, and Alicia Villanueva. Abstract Diagnosis for tccp using a Linear Temporal Logic. *Theory and Practice of Logic Programming*, 14(4-5):787–801, 2014. doi:10.1017/S1471068414000349.

13 Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A Fully Verified Executable LTL Model Checker. In *Computer Aided Verification*, pages 463–478. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-39799-8_31.

14 Joxe Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando Orejas. A Cut-Free and Invariant-Free Sequent Calculus for PLTL. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2007. doi:10.1007/978-3-540-74915-8_36.

15 Joxe Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando Orejas. Dual Systems of Tableaux and Sequents for PLTL. *The Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009. doi:10.1016/j.jlap.2009.05.001.

16 Rajeev Goré. Tableau Methods for Modal and Temporal Logics. In Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Springer Netherlands, Dordrecht, 1999. doi:10.1007/978-94-017-1754-0_6.

17 Alberto Griggio, Marco Roveri, and Stefano Tonetta. Certifying Proofs for LTL Model Checking. In *Formal Methods in Computer-Aided Design, FMCAD 2018, Austin, USA*, pages 1–9, October 2018. doi:10.23919/FMCAD.2018.8603022.

18 IEC. IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

19 ISO. Road vehicles – Functional safety, 2011.

20 Henry Kautz and Bart Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1194–1201. AAAI Press, 1996. URL: http://dl.acm.org/citation.cfm?id=1864519.1864564.

21 Jörg Kreiker, Andrzej Tarlecki, Moshe Y. Vardi, and Reinhard Wilhelm. Modeling, Analysis, and Verification - The Formal Methods Manifesto 2010 (Dagstuhl Perspectives Workshop 10482). *Dagstuhl Manifestos*, 1(1):21–40, 2011. doi:10.4230/DagMan.1.1.21.

22 Tuomas Kuismin and Keijo Heljanko. Increasing Confidence in Liveness Model Checking Results with Proofs. In Valeria Bertacco and Axel Legay, editors, *Hardware and Software: Verification and Testing*, pages 32–43. Springer International Publishing, 2013.

23 Orna Kupferman and Moshe Y. Vardi. From complementation to certification. *Theoretical Computer Science*, 345(1):83–100, 2005. doi:10.1007/978-3-540-24730-2_43.

24 Jianwen Li, Geguang Pu, Lijun Zhang, Moshe Y Vardi, and Jifeng He. Accelerating LTL satisfiability checking by SAT solvers. *Journal of Logic and Computation*, 28(6):1011–1030, April 2018. doi:10.1093/logcom/exy013.

25 Jianwen Li, Shufang Zhu, Geguang Pu, Lijun Zhang, and Moshe Y. Vardi. SAT-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods in System Design*, January 2019. doi:10.1007/s10703-018-00326-5.

26 Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, Berlin, Heidelberg, 1992. doi:10.1007/978-1-4612-0931-7.

27 Daniel Matichuk, Toby Murray, and Makarius Wenzel. Eisbach: A Proof Method Language for Isabelle. *Journal of Automated Reasoning*, 56, January 2016. doi:10.1007/s10817-015-9360-2.

28 Alain Mebsout and Cesare Tinelli. Proof Certificates for SMT-based Model Checkers for Infinite-state Systems. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*, FMCAD '16, pages 117–124, 2016. doi:10.1109/FMCAD.2016.7886669.

29 Kedar S. Namjoshi. Certifying Model Checkers. In *Proceedings of the 13th International Conference on Computer Aided Verification*, CAV '01, pages 2–13. Springer-Verlag, 2001. doi:10.1007/3-540-44585-4_2.

30 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. doi:10.1007/3-540-45949-9.

31 Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *International Symposium on Programming*, pages 337–351, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. doi:10.1007/3-540-11494-7_22.

32 Kristin Y. Rozier and Moshe Y. Vardi. LTL Satisfiability Checking. In Dragan Bošnački and Stefan Edelkamp, editors, *Model Checking Software*, pages 149–167, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-73370-6_11.

33 Aravinda P. Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 159–168, New York, NY, USA, 1982. ACM. doi:10.1145/800070.802189.